Non-Deterministic Finite Automata

In this section we extend our definition of deterministic finite automata to a seemingly more powerful notion, that of non-deterministic finite automata.

The surprising and wonderful result of this section is that these non-deterministic automata are actually not more powerful; they in fact describe the same set of languages.

Reading

Sections

Practice problems (page 83):

Motivation for non-deterministic automata

Finite automata have a certain rigidity to them: At every state and a given input, there is exactly one other state to transition to. This is precisely why they are called "deterministic".

But in so many practical situations we encounter non-determinism and are confronted with choices. A good example of this is trying to recognize the concatenation of two regular languages:

$$AB = \{wv \mid w \in A, v \in B\}$$

If we imagine a deterministic automaton trying to use the automata for A and B along the way, we could for instance imagine it starting with the automaton for A, then continuing with the automaton for B. It is this "continuing" part that is difficult: At what point should we drop A and start looking at B? How do we know this is the right time to do so?

To make this more concrete, suppose that the overall input is 1101001, and suppose that the words 11, 110 and 11010 are all valid words in A. Then that longer input may be in AB because 01001 is in B, or because 1001 is in B, or because 01 is in B, or maybe for all 3 reasons. But we can't know until we start looking into B. So after we have read the first two numbers following A's automaton, we have arrived at an accept state for A; do we continue or do we start looking into B? What if we do start at B and 0100 turns out not to be in B? We would conclude that the whole input isn't in AB (even though it could be there for other reasons).

So we have to make a choice at that point, and we don't know what the right choice would be, and we can't afford to make the wrong choice. So we can't make a choice. This is the problem presented by deterministic automata.

Definition of non-deterministic automata

The idea of non-deterministic automata is simple: We preserve the finite-ness and definite-ness of the states of a DFA, but we become more flexible on the transitions. From a state and on a given next input, you may now transition to 0 or more states. We also allow for "free transitions", called "epsilon-transitions", from a state to another without consuming any input. This way, at any given moment in the computation, our automaton might be in a variety/set of different states, not just one. And on each new input, the automaton would follow that input from all the different states it might have been in, resulting in a new list of possible states. When the computation ends, the automaton would possibly be in any number of possible states, and as long as one of these is an accepting state then the automaton would accept the string.

> A **(Non-deterministic) Finite Automaton** (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where: - Q is a finite set, called the *states*, - Σ is a finite set, called the *alphabet*, and we use Σ_ϵ to denote the alphabet extended with a new special symbol, ϵ, to indicate no use of input, - $\delta\colon Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the *transition function*, - $q_0 \in Q$ is the *start state*, - $F \subset Q$ is the set of *accept or final states* (possibly empty)

Here $\mathcal{P}(Q)$ denotes the power-set of the set Q. In other words the return values of the transition function are whole sets of states, instead of individual states.